



```

<HTML>
<HEAD>
<TITLE>00 Objects/Classes/Instances</TITLE>
</HEAD>
<BODY>

<P><FONT color="forestgreen" size="+2" id="arial">
  <B>Understanding Object Orientation Concepts</B>
</FONT></P>

```

```

<P><FONT color="black">
  <B>Objects & Classes</B>
</FONT></P>

```

```

<P>The world is full of <l>objects</l>. We naturally think of
objects in hierarchical categories, or <l>classes</l>. For
example, a computer is a general class of object. A hierarchy
of object classes surrounds the class &quot;computer&quot;;
extending in both directions. &quot;Computer&quot; is a
member of the more general class &quot;machines&quot;;. In
the other direction of the hierarchy are specific types of
computers: notebook computers, supercomputers, HP
computers, etc. If you are reading this document on your
computer, you are looking at an <l>instance</l> of the class
&quot;computer&quot;;.</P>

```

```

<P><CENTER>
  <IMG src="computer.gif" border="0">
</CENTER></P>

```

```

<HR />

```

```

Mo'00? Look up another concept:<BR>

```

```

<TABLE border="+2" width="+60%">
<TR>
  <TD>
    <A href="http://www.mo0o.org/inh.htm">Inheritance</A>
  <TD>
    <A href="http://www.mo0o.org/encap.htm">Encapsulation</A>
  <TD>
    <A href="http://www.mo0o.org/overld.htm">Overloading</A>
  </TD>
</TR>
</TABLE>

</BODY>
</HTML>

```

Fig. 2

```

main ()
{
310 {
    htmlDocument* document = new htmlDocument (stdout,
        "00 Objects/Classes/Instances");
    tableGrid*    table    = new tableGrid (1, 0, 0, "60%");
    centered*     center   = new centered ();

332 {
    document->add( new paragraph () );
    document->add( new htmlText ("Understanding Object Orientation
        Concepts", "forestgreen", normal, bold, "+2", "arial"));

334 {
    document->add( new paragraph () );
    document->add( new htmlText ("Objects & Classes", "black",
        normal, bold));

336 {
    explanation = query(oo_concept_database, concept);
    find_first_and_italicise(explanation_text, "object", "class",
        "instance");
    document->add( new paragraph() );
    document->add( new htmlText(explanation_text) );

340 {
    document->add( new paragraph() );
    center->add( new image(explanation_image) );
    document->add(center);

350 document->add( new horizontalRule() );

338 document->add( new htmlText("Mo' OO? Look up another link:") );

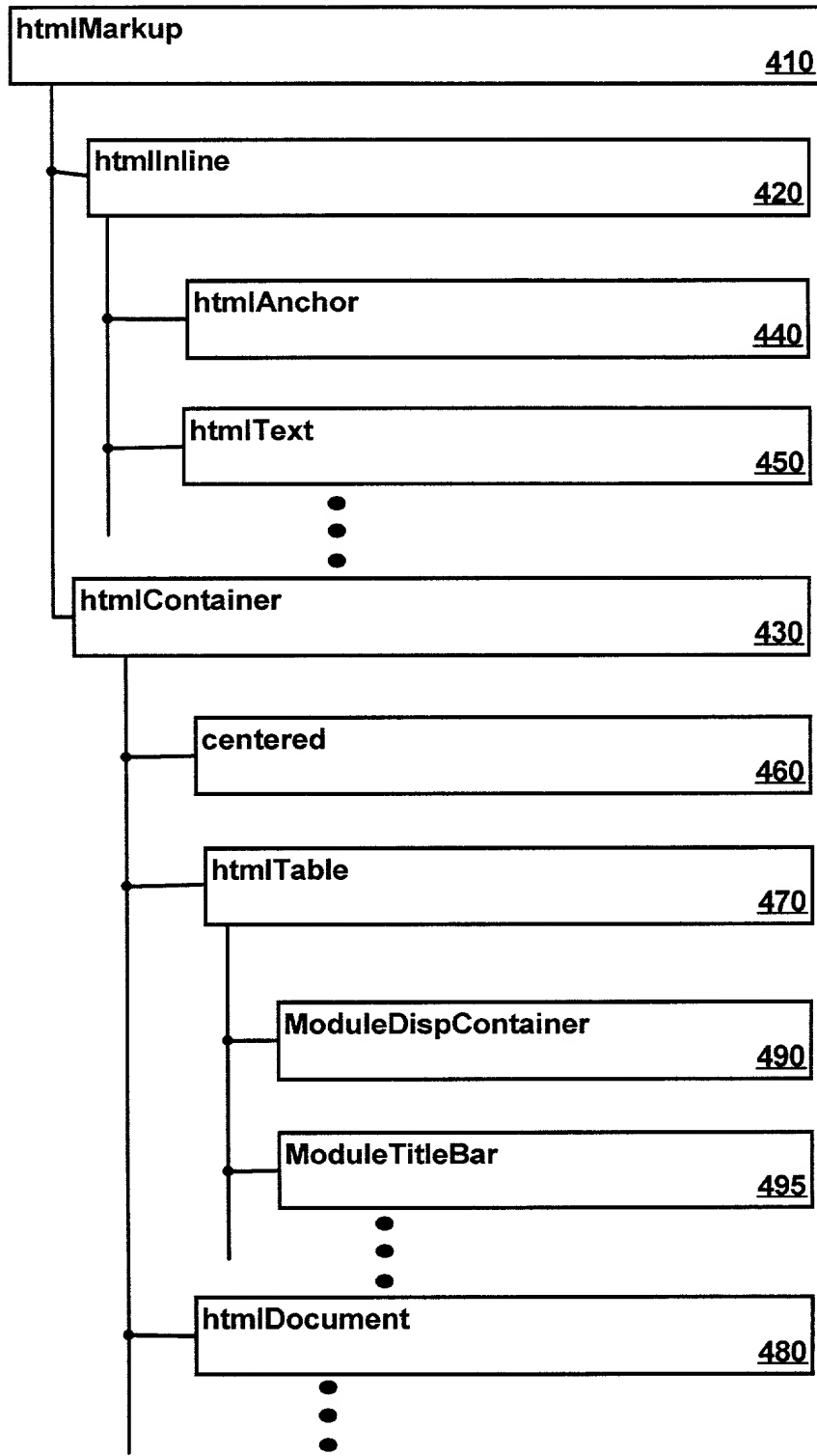
360 {
    table->newRow();
    table->addField( new anchor("http://www.mo00.com/Inheritance",
        new htmlText ("Inheritance") ) );
    table->addField( new anchor("http://www.mo00.com/Encapsulation",
        new htmlText ("Encapsulation") ) );
    table->addField( new anchor("http://www.mo00.com/Overloading",
        new htmlText ("Overloading") ) );
    document->add(table);

    delete document;
}
}

```

Fig. 3

400



*Fig. 4*

410

// This class is an interface for defining the basic HTML/XML  
// relationship between a child element and its parent.

```
class htmlMarkup
{
protected:
    htmlMarkup* parent = NULL;
    FILE*      fptr    = NULL;
public:
    htmlMarkup();
    virtual ~htmlMarkup();
    virtual setParent(htmlMarkup* parent) ~ 510
        { this.parent = parent }
}
```

*Fig. 5*

420

```
class htmlInline extends htmlMarkup
{
protected:
    DynamicArray* buffer = NULL;
public:
    htmlInline();
    virtual ~htmlInline()
        { if (buffer) fprintf(parent.fptr, "%s", buffer)} ~ 610
}
```

*Fig. 6*

440

```
class htmlAnchor extends htmlInline
{
public:
    htmlAnchor (String href, htmlMarkup* label) ~ 710
        { buffer  = "<a";
          buffer += " href="+href;
          buffer += ">";

          // flush the label markup to this buffer
          label.setParent(this);
          delete label;
          buffer += "</a>";
        }
}
```

*Fig. 7*

430

```
class htmlContainer extends htmlMarkup
{
protected:
    FILE*fptr = NULL;
public:
    htmlContainer();
    virtual ~htmlContainer()
        { if (fptr && parent.fptr)
            concatenateFiles(fptr, parent.fptr); }
}
```

*Fig. 8*

450

```
class htmlTable extends htmlContainer
{
public:
    htmlTable()
        { fptr = new temporaryFile();
          print("<table>");
        }

    virtual ~htmlTable ()
        { print("</table>"); }

    void addRow()
        { print("<tr>");    }

    void addContent(htmlMarkup* content)
        { print("<td>");

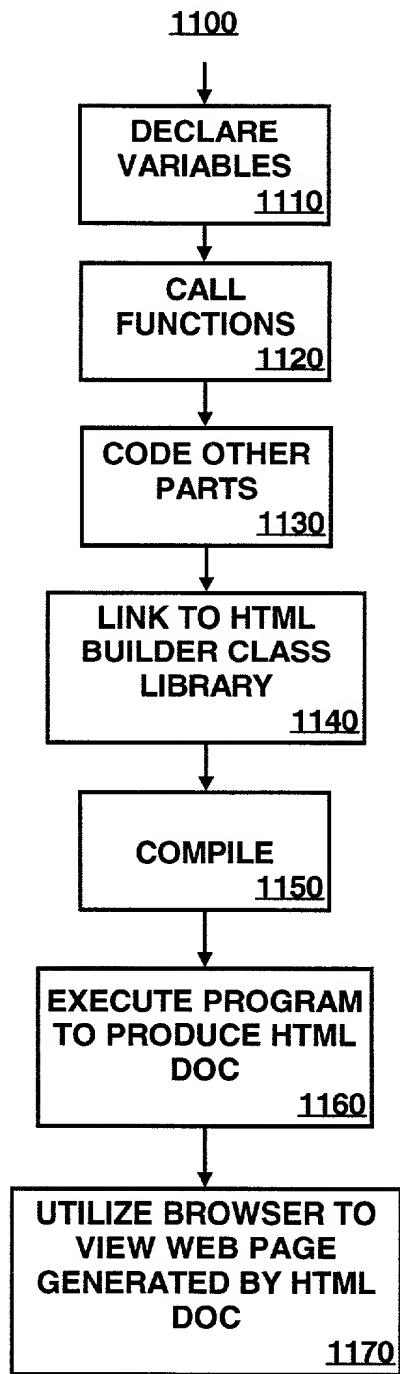
          // flush the child content to this table
          content.setParent(this);
          delete content;

          print("</td>");
        }
}
```

*Fig. 9*

Class	Style	HTML element
commentText	htmlInline	<!-- -- >
htmlText	htmlInline	ASCII text
formattedText	htmlInline	<PRE>
embeddedText	htmlInline	<LAYER>
htmlImage	htmlInline	<IMG>
htmlAnchor	htmlInline	<A>
paragraph	htmlInline	<P>
centered	htmlContainer	<CENTER>
lineBreak	htmlInline	 
noLineBreak	htmlInline	<NOBR>
horizontalRule	htmlInline	<HR>
table	htmlContainer	<TABLE>
htmlDocument	htmlContainer	<HTML>
htmlForm	htmlContainer	<FORM>
formInput	htmlInline	<INPUT>
formTextReadOnly	htmlInline	<TEXT>
selectionList	htmlContainer	<SELECTION>

Fig. 10



*Fig. 11*